2. Aufgabenblatt zu Funktionale Programmierung vom Mo, 24.10.2016. Fällig: Mi, 02.11.2016 (15:00 Uhr)

Themen: *Funktionen über ganzen Zahlen, Wahrheitswerten, Listen und Tupeln*

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften für die Lösung der unten angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `Aufgabe2.hs` ablegen. Sie sollen für die Lösung dieses Aufgabenblatts also ein "gewöhnliches" Haskell-Skript schreiben. Versehen Sie wieder alle Funktionen, die Sie zur Lösung brauchen, mit ihren Typdeklarationen und kommentieren Sie Ihre Programme aussagekräftig. Benutzen Sie, wo sinnvoll, Hilfsfunktionen und Konstanten.

Zur Frist der Zweitabgabe: Siehe allgemeine Hinweise auf Aufgabenblatt 1.

1. Wir betrachten noch einmal die Fakultätsfunktion:

$$!: IN \to IN$$
$$n! = \begin{cases} 1 & \text{falls } n = 0 \\ n * (n-1)! & \text{sonst} \end{cases}$$

   - Schreiben Sie eine Haskell-Rechenvorschrift `facLst` mit der Signatur `facLst :: Integer -> [Integer]`.
     Angewendet auf ein Argument $n \geq 0$ liefert `facLst` eine aufsteigend geordnete Liste der Werte der Fakultätsfunktion für die Werte von $0, 1, 2, \ldots, n$.
     Ist $n < 0$, so liefert `facLst` die leere Liste als Resultat.
   - Schreiben Sie eine Variante `factsL` mit der Signatur `factsL :: Integer -> [Integer]`.
     Angewendet auf ein Argument $n \geq 0$ liefert `factsL` eine absteigend geordnete Liste der Werte der Fakultätsfunktion für die Werte von $0, 1, 2, \ldots, n$.
     Ist $n < 0$, so liefert `factsL` die leere Liste als Resultat.

   Anwendungsbeispiele:

```
facLst 5 ->> [1,1,2,6,24,120]
facLst (-5) ->> []
factsL 5 ->> [120,24,6,2,1,1]
factsL (-5) ->> []
```

2. In der Zeichenreihe `"a16B008Lk1234n1151248cvK"` sind Teilzeichenfolgen jeweils maximaler Länge enthalten, die ausschließlich aus Ziffern bestehen, nämlich die Teilzeichenreihen

   `"16"`,`"008"`,`"1234"` und `"1151248"`,

   sog. *Numerale maximaler Länge*.

   Schreiben Sie eine Haskell-Rechenvorschrift `extractNumerals` mit der Signatur

```
extractNumerals :: String -> [String]
```

   die angewendet auf eine Zeichenreiche s eine (möglicherweise leere) Liste der in s enthaltenen Numerale maximaler Länge von s liefert. Die Numerale sollen im Resultat von `extractNumerals` dabei in der gleichen Weise von links nach rechts angeordnet sein wie es ihren Vorkommen in s entspricht.

   Anwendungsbeispiele:

```
extractNumerals "a16B008Lk1234n1151248cvK" ->> ["16","008","1234","1151248"]
extractNumerals "abcDEFghi" ->> []
```

3. Schreiben Sie zwei Haskell-Rechenvorschriften `isPowOf2` ("isPowerOf2") und `sL2pO2` ("StringList2PowerOf2")

```
isPowOf2 :: Int -> (Bool,Int)
sL2pO2 :: [String] -> [Int]
```

mit folgenden Funktionalitäten:

- Angewendet auf eine Zahl n, $n \geq 0$, liefert die Funktion `isPowOf2` das Resultat `(True,m)`, falls gilt: $n = 2^m$. Ist $n < 0$ oder keine Potenz von 2, ist das Resultat von `isPowOf2` das Paar `(False,-1)`.
- Angewendet auf eine Liste $s = [s_1, ..., s_k]$ von Zeichenreihen, liefert die Funktion `sL2pO2` eine gleich lange Liste $t = [m_1, ..., m_k]$ ganzer Zahlen. Dabei gilt für alle $1 \leq i \leq k$: Ist s_i ein Numeral, dessen Dezimalwert eine positive Zweierpotenz ist, so ist m_i der Exponent dieser Zweierpotenz. Ist s_i kein Numeral oder kein Numeral mit dieser Eigenschaft, so ist m_i gleich -1.

Anwendungsbeispiele:

```
isPowOf2 8 ->> (True,3)
isPowOf2 6 ->> (False,-1)

sL2pO2 ["32","007","008","1024","31"] ->> [5,-1,3,10,-1]
sL2pO2 (extractNumerals "a16B008Lk1234n1151248cvK") ->> [4,3,-1,-1]
sL2pO2 (extractNumerals "abcDEFghi") ->> []
```

4. Schreiben Sie in Analogie zu den Funktionalen `curry` und `uncurry` aus der Vorlesung entsprechende Funktionale

- `curry3 :: ((a,b,c) -> d) -> a -> b -> c -> d`
- `uncurry3 :: (a -> b -> c -> d) -> (a,b,c) -> d`

und

- `curry4 :: ((a,b,c,d) -> e) -> a -> b -> c -> d -> e`
- `uncurry4 :: (a -> b -> c -> d -> e) -> (a,b,c,d) -> e`

Überlegen Sie sich geeignete Funktionen, um die korrekte Arbeitsweise der einzelnen Funktionale zu überprüfen und testen. Implementieren Sie sie und testen Sie Ihre Implementierungen der Funktionale damit.

Wichtig: Wenn Sie einzelne Rechenvorschriften aus früheren Lösungen für dieses oder spätere Aufgabenblätter wieder verwenden möchten, so kopieren Sie diese in die neue Abgabedatei ein. Ein `import` schlägt für die Auswertung durch das Abgabeskript fehl, weil Ihre alte Lösung, aus der importiert wird, nicht mit abgesammelt wird. Deshalb: Kopieren statt importieren zur Wiederwendung!

Haskell Live

Am Freitag, den 28.10.2016, oder an einem der späteren Termine werden wir uns in *Haskell Live* u.a. mit der Aufgabe *"Krypto Kracker!"* beschäftigen.

Krypto Kracker!

Eine ebenso populäre wie einfache und unsichere Methode zur Verschlüsselung von Texten besteht darin, eine Permutation des Alphabets zu verwenden. Bei dieser Methode wird jeder Buchstabe des Alphabets einheitlich durch einen anderen Buchstaben ersetzt, wobei keine zwei Buchstaben durch denselben Buchstaben ersetzt werden. Das stellt sicher, dass verschlüsselte Texte auch wieder eindeutig entschlüsselt werden können.

Eine Standardmethode zur Entschlüsselung nach obiger Methode verschlüsselter Texte ist als "reiner Textangriff" bekannt. Diese Angriffsmethode beruht darauf, dass der Angreifer den Klartext einer Textphrase kennt, von der er weiß, dass sie in verschlüsselter Form im Geheimtext vorkommt. Durch den Vergleich von Klartext- und verschlüsselter Phrase wird auf die Verschlüsselung geschlossen, d.h. auf die verwendete Permutation des Alphabets. In unserem Fall wissen wir, dass der Geheimtext die Verschlüsselung der Klartextphrase

`the quick brown fox jumps over the lazy dog`

enthält.

Ihre Aufgabe ist nun, eine Liste von Geheimtextphrasen, von denen eine die obige Klartextphrase codiert, zu entschlüsseln und die entsprechenden Klartextphrasen auszugeben. Kommt mehr als eine Geheimtextphrase als Verschlüsselung obiger Klartextphrase in Frage, geben Sie alle möglichen Entschlüsselungen der Geheimtextphrasen an. Im Geheimtext kommen dabei neben Leerzeichen ausschließlich Kleinbuchstaben vor, also weder Ziffern noch sonstige Sonderzeichen.

Schreiben Sie ein Programm in Haskell oder in irgendeiner anderen Programmiersprache ihrer Wahl, das diese Entschlüsselung für eine Liste von Geheimtextphrasen vornimmt.

Angewendet auf den aus drei Geheimtextphrasen bestehenden Geheimtext (der in Form einer Haskell-Liste von Zeichenreihen vorliegt)

```
["vtz ud xnm xugm itr pyy jttk gmv xt otgm xt xnm puk ti xnm fprxq",
 "xnm ceuob lrtzv ita hegfd tsmr xnm ypwq ktj",
 "frtjrpgguvj otvxmdxd prm iev prmvx xnmq"]
```

sollte Ihre Entschlüsselungsfunktion folgende Klartextphrasen liefern (ebenfalls wieder in Form einer Haskell-Liste von Zeichenreihen):

```
["now is the time for all good men to come to the aid of the party",
 "the quick brown fox jumps over the lazy dog",
 "programming contests are fun arent they"]
```